ASSIGNMENT NO.4

Title: Build the Image classification model

Aim: Build the Image classification model by dividing the model into following 4 stages:
Loading and pre-processing the image data
Defining the model's architecture
Training the model
Estimating the model's performance

Theory:

Since the vast amount of image data we obtain from cameras and sensors is unstructured, we depend on advanced techniques such as machine learning algorithms to analyze the images efficiently. Image classification is probably the most important part of digital image analysis. It uses AI-based deep learning models to analyze images with results that for specific tasks already surpass human-level accuracy (for example, in face recognition).

Performing machine learning for image recognition at the edge makes it possible to overcome the limitations of the cloud in terms of privacy, real-time performance, efficacy, robustness, and more. Hence, the use of Edge AI for computer vision makes it possible to scale image recognition applications in real-world scenarios.

Image classification is a supervised learning problem: define a set of target classes (objects to identify in images), and train a model to recognize them using labeled example photos. Image classification is the task of classifying and assigning labels to groupings of images or vectors within an image, based on certain criteria. A label can be assigned based on one or more criteria.

It's a fundamental problem in the field of machine learning and computer vision and has a wide range of applications, including:

- Object Recognition: Identifying objects within an image, such as detecting cars in traffic, recognizing animals in wildlife photos, or identifying food items in pictures.
- Facial Recognition: Determining whether a face is present in an image and, if so, classifying whose face it is, which is used in applications like security systems and social media tagging.
- Disease Diagnosis: Diagnosing medical conditions based on medical images like Xrays, MRIs, or histopathology slides.

This process typically involves collecting and preprocessing labeled image data, training deep learning models like convolutional neural networks (CNNs), and evaluating their performance using metrics like accuracy and F1-score.

The process of image classification typically involves the following steps:

- Data Collection,
- Data Preprocessing,
- Feature Extraction,
- Model Training,
- Model Evaluation,
- Deployment

Deep learning for Image classification

Deep learning excels in recognizing objects in images as it's implemented using 3 or more layers of artificial neural networks where each layer is responsible for extracting one or more feature of the image. Computers are able to perform computations on numbers and is unable to interpret images in the way that we do. We have to somehow convert the images to numbers for the computer to understand.

Deep Learning makes use of Neural Networks. Data is transmitted between nodes (like neurons in the human brain) using complex, multi-layered neural connections. Each of these nodes processes the data and relays the findings to the next tier of nodes. As a response, the data undergoes a non-linear modification that becomes progressively abstract. Various kinds of Neural Networks exist depending on how the hidden layers function. For example, Convolutional Neural Networks, or CNNs, are commonly used in Deep Learning image classification.

In this type of Neural Network, the output of the nodes in the hidden layers of CNNs is not always shared with every node in the following layer. It's especially useful for image processing and object identification algorithms.

Here are some reasons for using deep learning in image classification:

1. **Feature Learning**: Deep learning models, particularly convolutional neural networks (CNNs), are capable of automatically learning relevant features from the raw pixel values of images. This eliminates the need for manual feature engineering, where engineers would traditionally design features to represent objects or patterns in images.
2. **Hierarchical Representations**: Deep networks learn hierarchical representations of features. Lower layers capture simple features like edges and textures, while higher layers learn more complex features and object representations. This hierarchical approach is wellsuited for the diverse and intricate nature of images.

3. **Scalability:** Deep learning models can scale with more data and larger architectures. This allows them to capture intricate patterns and generalize well to a wide variety of images.
4. **State-of-the-Art Performance**: Deep learning models, especially CNNs, have consistently achieved state-of-the-art performance on image classification benchmarks.
5. **Transfer Learning**: Pre-trained deep learning models can be fine-tuned for specific image classification tasks, reducing the amount of labelled data and training time required.

A convolutional neural network (CNN or convnet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice. This makes them highly suitable for computer vision (CV) tasks and for applications where object recognition is vital, such as self-driving cars and facial recognition. The Convolutional layer applies filters to the input image to extract features, the Pooling layer down samples the image to reduce computation, and the fully connected layer makes the final prediction. Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

- Convolutional layer
- Pooling layer
- Fully-connected (FC) layer

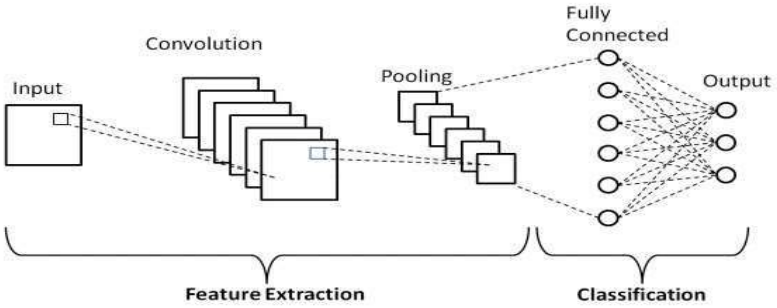

Figure 2: CNN Architecture

Types of convolutional neural networks

- AlexNet
- VGGNet
- GoogLeNet

- ResNet
- ZFNet

Applications

Below are some applications of Convolutional Neural Networks:

1.Object detection

2. Semantic segmentation.

3. Image captioning

Convolution Operation

The primary purpose of the convolution operation is to extract local features from the input data. By using different kernels, the network can learn to recognize specific patterns or features, such as edges, corners, textures, or more complex structures. The convolution operation uses a sliding window or kernel to traverse the entire input image. The kernel is typically much smaller than the input image but covers a local region. At each position of the kernel, a dot product is computed with the portion of the image under the kernel.

Strides: The step size by which the kernel moves is controlled by the stride parameter. A larger stride results in a downsized output, while a smaller stride leads to a higher-resolution output.

Convolution Kernels

A kernel is a small 2D matrix whose contents are based upon the operations to be performed. Kernels are learnable parameters in a CNN, and the network adapts them during the training process to capture specific features. A kernel maps on the input image by simple matrix multiplication and addition, the output obtained is of lower dimensions and therefore easier to work with. Each element in the kernel has a weight value. During the convolution operation, the kernel's weights are applied to the corresponding pixel values in the input image. The dot product of the kernel and the local image region is computed to produce an element in the output feature map.

The matrix of weights is referred to as the Kernel or Filter. In this case, we have the kernel of size 2X2.

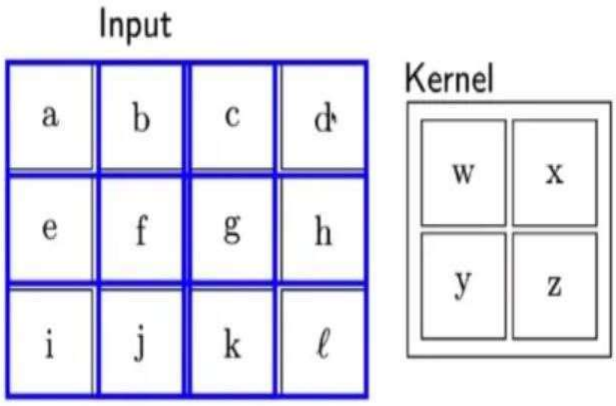

Figure 3: Convolutional Kernels

The shape of a kernel is heavily dependent on the input shape of the image and architecture of the entire network, mostly the size of kernels is (MxM) i.e., a square matrix. The movement of a kernel is always from left to right and top to bottom.

Stride defines by what step does to kernel move, for example stride of 1 makes kernel slide by one row/column at a time and stride of 2 moves kernel by 2 rows/columns.

For input images with 3 or more channels such as RGB a filter is applied.

Filters are one dimension higher than kernels and can be seen as multiple kernels stacked on each other where every kernel is for a particular channel.

Therefore, for an RGB image of (32x32) we have a filter of the shape say (5x5x3).

For a 3D input matrix, the movement of the kernel will be from front to back, left to right and top to bottom. By now you must have a basic understanding of the Convolution operation which is the essence of a Convolutional Neural Network.

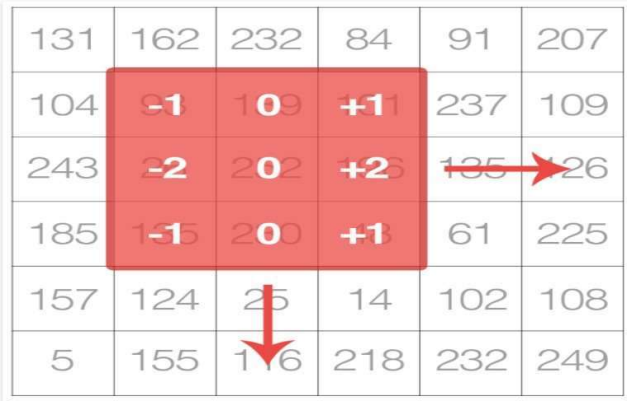

Figure 5: Convolutional Kernels

A **kernel** is a small matrix that slides across from left-to-right and top-to-bottom of a larger image. At each pixel in the input image, the neighbourhood of the image is convolved with the kernel and the output stored.

Kernels can be an arbitrary size of M x N pixels, provided that both M and N are odd integers, to always ensure there is a valid (x, y)-coordinate at the centre of the kernel.

In image processing, convolution requires 3 components:

1. An input image.
2. A kernel matrix that we are going to apply to the input image.
3. An output image to store the output of the input image convolved with the kernel.

Types of Convolutions:

Convolution Type	**Description**	**Use Cases**
Standard Convolution	Basic convolution operation using a kernel to detect features in input data.	General feature extraction
Dilated (Atrous) Convolution	Introduces gaps between kernel elements to increase receptive field.	Multi-scale feature detection
Depth-wise Separable Convolution	Splits convolution into depth-wise and pointwise convolutions to reduce parameters.	Mobile and resource-constrained models
Grouped Convolution	Applies different kernels to input channels in groups.	Improving efficiency and parallelism
Separable Convolution	Combines depthwise convolution with pointwise convolution.	Reducing parameters and computation

Convolution Layers:

A deep learning CNN consists of three layers: a convolutional layer, a pooling layer and a fully connected (FC) layer. The convolutional layer is the first layer while the FC layer is the last. From the convolutional layer to the FC layer, the complexity of the CNN increases.

Convolutional layer: The majority of computations happen in the convolutional layer, which is the core building block of a CNN. A second convolutional layer can follow the initial convolutional layer. The process of convolution involves a kernel or filter inside this layer moving across the receptive fields of the image, checking if a feature is present in the image. After each iteration a dot product is calculated between the input pixels and the filter. The final output from the series of dots is known as a feature map.

Pooling layer. Like the convolutional layer, the pooling layer also sweeps a kernel or filter across the input image. But unlike the convolutional layer, the pooling layer reduces the number of parameters in the input and also results in some information loss. On the positive side, this layer reduces complexity and improves the efficiency of the CNN.

Fully connected layer: The FC layer is where image classification happens in the CNN based on the features extracted in the previous layers. Here, fully connected means that all the inputs or nodes from one layer are connected to every activation unit or node of the next layer.

Feature extraction in neural networks contains the representations that are learned by the previous network to extract the interesting features from new samples. The features are then run through the new classifier which is already trained from scratch. Feature extraction, in case of the convents, consists of taking convolutional base of the previously trained network then running a new data through it and finally training the new classifier on top of output of the network.

The feature maps of the convnet are the presence maps of generic concepts over a picture which is useful regardless of the computer-vision problem or any other problem.

Here's how feature extraction is done with convolution layers:

1. Local Receptive Fields:

- In the first layer of a CNN, each neuron focuses on a small region of the input image, known as its local receptive field.
- A convolution layer applies a set of learnable filters (kernels) to these local receptive fields.

These filters are small matrices, typically 3x3 or 5x5, which slide across the input.

2. **Convolution Operation**:

- The convolution operation is performed by sliding these filters (kernels) over the input image, one local receptive field at a time.
- At each position, the filter is applied to the local receptive field, and a weighted sum of the elements is computed. This operation captures patterns or features in the local region.

3. **Feature Map Creation:**

- The result of each convolution operation at different positions forms a feature map. Each feature map represents a particular feature or pattern detected by the corresponding filter.
- By using multiple filters in a single convolution layer, the network can learn to detect various features at different scales and orientations.

4. **Non-linearity (Activation Function):**

- After the convolution operation, an activation function (commonly ReLU - Rectified Linear Unit) is applied element-wise to the feature map.
- The activation function introduces non-linearity, allowing the network to capture complex relationships between features.

5. **Downsampling (Pooling):**

- In some CNN architectures, after a series of convolution and activation operations, downsampling is applied through pooling layers (e.g., max-pooling or average-pooling).
- Pooling reduces the spatial dimensions of the feature maps, making the network more computationally efficient while preserving important features.

Steps/ Algorithm

1. Choose a dataset of your interest or you can also create your own image dataset (Ref : https://www.kaggle.com/datasets/) Import all necessary files.

(Ref : https://www.analyticsvidhya.com/blog/2021/01/image-classification-usingconvolutional- neural-networks-a-step-by-step-guide/)

Libraries and functions required Tensorflow,keras

numpy : NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. To import numpy use import numpy as np

pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. To import pandas use import pandas as pd

sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. For importing train_test_ split use

Prepare Dataset for Training : //Preparing our dataset for training will involve assigning paths and creating categories(labels), resizing our images.

Create a Training a Data : // Training is an array that will contain image pixel values and the index at which the image in the CATEGORIES list.

Shuffle the Dataset

Assigning Labels and Features

Normalising X and converting labels to categorical data

Split X and Y for use in CNN

Define, compile and train the CNN Model Accuracy

and Score of model.

INPUT CODE:

```
import numpy as np import pandas as pd import random import tensorflow
as tf import matplotlib.pyplot as plt from sklearn.metrics import
accuracy_score from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Flatten, Conv2D, Dense, MaxPooling2D from
tensorflow.keras.optimizers import SGD from tensorflow.keras.utils import
to_categorical from tensorflow.keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()

print(X_train.shape)

X_train[0].min(), X_train[0].max()
X_train = (X_train - 0.0) / (255.0 - 0.0)
X_test = (X_test - 0.0) / (255.0 - 0.0)
X_train[0].min(), X_train[0].max()

def plot_digit(image, digit, plt, i):
    plt.subplot(4, 5, i + 1)
    plt.imshow(image, cmap=plt.get_cmap('gray'))
plt.title(f'Digit: {digit}")     plt.xticks([])
plt.yticks([]) plt.figure(figsize=(16, 10)) for i in
range(20):
    plot_digit(X_train[i], y_train[i], plt, i)    plt.show()

X_train = X_train.reshape((X_train.shape + (1,)))
X_test = X_test.reshape((X_test.shape + (1,)))

y_train[0:20]

model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
```

```
    Dense(100, activation="relu"),
    Dense(10, activation="softmax")
])

optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(      optimizer=optimizer,
loss="sparse_categorical_crossentropy",
metrics=["accuracy"]) model.summary()

model.fit(X_train, y_train, epochs=10, batch_size=32)

plt.figure(figsize=(16, 10)) for
i in range(20):
    image = random.choice(X_test).squeeze()
    digit = np.argmax(model.predict(image.reshape((1, 28, 28, 1)))[0], axis=-1)
plot_digit(image, digit, plt, i) plt.show() predictions =
np.argmax(model.predict(X_test), axis=-1) accuracy_score(y_test,
predictions)

n=random.randint(0,9999)
plt.imshow(X_test[n]) plt.show()
predicted_value=model.predict(
X_test) print("Handwritten
number in the image is= %d"
%np.argmax(predicted_value[n]
))

score = model.evaluate(X_test, y_test, verbose=0) print('Test
loss:', score[0]) #Test loss: 0.0296396646054 print('Test
accuracy:', score[1])

#The implemented CNN model is giving Loss=0.040644191205501556 and
#accuracy: 0.9878000020980835 for test MNIST dataset
```

OUTPUT

```
In [2]:
1 (X_train, y_train), (X_test, y_test) = mnist.load_data()

In [3]:
1 print(X_train.shape)

(60000, 28, 28)

In [4]:
1 X_train[0].min(), X_train[0].max()

Out[4]: (0, 255)

In [5]:
1 X_train = (X_train - 0.0) / (255.0 - 0.0)
2 X_test = (X_test - 0.0) / (255.0 - 0.0)
3 X_train[0].min(), X_train[0].max()

Out[5]: (0.0, 1.0)
```

Figure 1: Output 1

The above figure shows the training of model.

```
In [12]:
1 model = Sequential([
2     Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
3     MaxPooling2D((2, 2)),
4     Flatten(),
5     Dense(100, activation="relu"),
6     Dense(10, activation="softmax")
7 ])

In [13]:
1 optimizer = SGD(learning_rate=0.01, momentum=0.9)
2 model.compile(
3     optimizer=optimizer,
4     loss="sparse_categorical_crossentropy",
5     metrics=["accuracy"]
6 )
7 model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)
```

Figure 2: Output 2

The above figure shows defining of model.

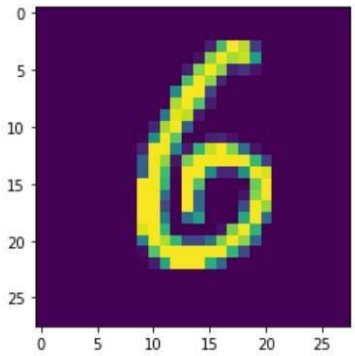

Figure 3: Prediction of digit

The above figure shows the prediction of the digit written.

```
In [18]:
1 predicted_value=model.predict(X_test)
2 print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]))

313/313 [==============================] - 2s 6ms/step
Handwritten number in the image is= 6

In [19]:
1 score = model.evaluate(X_test, y_test, verbose=0)
2 print('Test loss:', score[0]) #Test loss: 0.0296396646054
3 print('Test accuracy:', score[1])

Test loss: 0.040644191205501556
Test accuracy: 0.9878000020980835

In [ ]:
1 #The implemented CNN model is giving Loss=0.040644191205501556  and
2 #accuracy: 0.9878000020980835 for test mnist dataset
```

Figure 4: Implemented model

The above figure shows the Loss, Accuracy, and prediction of the model.

Conclusion:

Thus, using deep learning algorithm we have implemented and tested the model for Image Classification.

After training and testing, the accuracy of the model for Image Classification is 98.7% and loss 04.06%.